DI コンテナによるパターンの漸増的な学習

太田 健一郎†

企業のシステム開発において, テスト容易性, 変更容易性, 保守性等の観点からソフトウェア・パターンの適切な利用が求められている. しかし, 現状では学習の難易度, 効果の不透明性等から開発メンバーの一部が学習, 利用しているに過ぎない. 筆者はその解決策の一つとして昨今, 注目されている DI(Dependency Injection)コンテナの利用を提唱する. この DI コンテナを土台として, システム設計をすることによって, 開発メンバーが, ソフトウェア・パターンに対し, 大きく構えることなく, 自然な形で学び, 利用できるようになることを示す.

Progressive study of the patterns by DI container

KENICHIROH OHTA†

In the system developments of companies, from viewpoints, such as test ease, change ease, and conservativeness, suitable use of software patterns is called for. But in the present condition, some development members have learned and adapted themselves from the difficulty of study, and the opacity of effect etc. I consider DI(Dependency Injection) container which attracts attention now as one of the solution. I will show that it can learn in a natural form without a development member posing greatly to software patterns by using this DI container as a foundation.

1. はじめに

企業のシステム開発において, テスト容易性, 変更容易性, 保守性等の観点からソフトウェア・パターンの適切な利用が求められている. 適切なソフトウェア・パターンの利用は, メンバーの分業化, テストの容易性, 変更の容易性, 保守の容易性等をもたらし, プロジェクトを成功に導く重要な要素となる.

しかし, 下記の理由から, 比較的ソフトウェア・パターンに造詣が深いアーキテクチャチームのメンバーを除いて, 各メンバーが適切にパターンを利用することは難しい.

- ビジネス・ロジックを設計する設計者は対象ドメインについては詳しいが, ソフトウェア・パターンに興味を示し, 造詣が深い人間は少ない
- ビジネス・ロジックの実装者であるプログラマの中にはソフトウェア・パターンについて造詣が深く, 適切に利用できる人間もいる. しかし, 平均的に見るとプログラマの多くはイディオムやコーディングテクニックについては詳しいが, デザイン・パターン以上の抽象度のパターンの学習, 利用にそれほど必然性を感じなかったり, あまりに多くのパターンに圧倒されて学ぶ意欲がそがれてしまったりしている場合が多く, 結果的に個別最適にパターンを用いるようになってしまっている

2. 最小限のパターンの学習

上記の問題を解消し, 各メンバーが適切にパターンを利用できるようにするには, 以下のようなことが実現できれば良い.

- メンバーにとってパターンの学習と利用が有用なものであることを示すこと
- メンバーが最小限度の学習で効果をもたらすような最小セットのパターンを提示すること
- メンバーが最小セットのパターンを足がかりにして, もっと広く深く有用なパターンを学習し, 利用できるようにすること
- メンバーが最小セットのパターンを構えることなく学習できるような仕組みを提供すること

3. オブジェクト指向設計の五原則と DI コンテナ

上記を実現するための最小限のパターンが文献[1]で解説しているオブジェクト指向設計の五原則であり,

†日本アイ・ビー・エム株式会社
IBM Japan

パターンを学ぶ仕組みとして有用なのが，次世代の J2EE の技術として注目されている DI コンテナである．それぞれについて，概要と利用効果を簡単に説明する．

3.1. オブジェクト指向設計の五原則

文献[1]で解説しているオブジェクト指向設計の五原則は，変更容易性，分析・設計・実装のシームレス化，テスト容易性等のオブジェクト指向設計のメリットを実現するために最低限度，開発者が知っておくべき設計の原則である．GoF のデザイン・パターン等，各種のデザイン・パターンはこの五原則を土台としているため，この五原則を理解していれば，各種のデザイン・パターンの理解と利用も容易になる．

五原則は以下の通りである．クラス図等を交えた詳細な解説については，文献[1]を参考にして欲しい．

- 単一責任の原則
- オープン・クローズドの原則
- リスコフの置換原則
- 依存関係逆転の原則
- インタフェース分離の法則

3.2. DI コンテナ

次世代の J2EE の中核として注目されているのが，DI コンテナである．DI コンテナには Spring[2]，Seasar[3]といったものがあり，同時に AOP(Aspect Oriented Programming)の概念もサポートしていることが多い．DI コンテナの提供する機能の概要は以下の通りである．

DI コンテナ: オブジェクトの生成，依存関係を一括管理する仕組みを提供する

AOP: システムの中核機能と横断的機能を分離し，実行時に結合する仕組みを提供する

Spring や Seasar といった DI コンテナの提唱者は J2EE を初めとする 3 層型システムについて，DI コンテナを活かした設計手法も同時に提唱しており，DI コンテナをプラットフォームにすると同時に彼らの設計手法を実践することにより，先ほど述べたオブジェクト指向の五原則を満たすことが可能となる．DI コンテナの利用と設計の効果を以下に示す．

各クラスの責任を明確化し，AOP で横断的機能を分離

「単一責任の原則」が満たされる

インタフェースに対するプログラミングとコンテナによる依存関係解決

「オープン・クローズドの原則」が満たされる

「リスコフの置換原則」が満たされる

「インタフェース分離の原則」が満たされる

明確なレイヤリングとインタフェースに対するプログラミング

「依存関係逆転の原則」が満たされる

高度な設計能力を持った技術者でなくても，オブジェクト指向設計の基本原則を満たした設計が可能になる

変更容易性とテスト容易性が高い設計になる

DI コンテナの利用とそれを活かした設計手法を実践することにより，オブジェクト指向設計の五原則を満たし，設計のどの部分に原則とパターンが利用されているかを容易に学習し，利用効果を理解できるようになる．

4. 結論

システム開発において，ソフトウェア・パターンを適切に利用し，メンバーの学習と実践に役立つ方法として，オブジェクト指向設計の五原則と DI コンテナの利用を説明した．これによって，メンバーのパターンの学習と利用において，以下のような効果が期待できる．

- メンバーはオブジェクト指向設計の五原則を適応することによって，パターンの学習と利用が有用であることを理解できるようになる
- メンバーがパターンの利用に対して，構えることなくおのずからパターンを利用しているという状況を作り出せる
- メンバーがオブジェクト指向設計の五原則を足がかりにして，自分や別のドメインの有用なパターンを学習し，利用できるようになる

メンバーが自然な形で最小限のパターンを学習，利用できるような環境を整えることによって，長期的に見た場合，システム開発に携わるメンバーの多くが適切にパターンを利用し，パターンの利用がシステムの品質とプロジェクトの成功に寄与できるようになる．

上記を踏まえてワークショップでは，システム開発におけるコミュニケーションと学習のためのパターンの活用方法について議論したい．

参考文献

[1] ロバート・C・マーチン，アジャイルソフトウェア開発の奥義，ソフトバンクパブリッシング，2004

[2] Spring, http://www.springframework.org/

[3] Seasar, http://www.seasar.org/